Two-stage Cascaded Classifier for Purchase Prediction

Sheikh Muhammad Sarwar
University of Dhaka, Bangladesh
smsarwar@du.ac.bd (DU_Erudite team)

Mahamudul Hasan
University of Dhaka, Bangladesh
munna09bd@gmail.com

Dmitry I. Ignatov
National Research University
Higher School of Economics, Moscow, Russia
dignatov@hse.ru

ABSTRACT

In this paper we describe our machine learning solution for the RecSys Challenge 2015. We have proposed a time-efficient two-stage cascaded classifier for the prediction of buy sessions and purchased items within such sessions. Based on the model, several interesting features found, and formation of our own test bed, we have achieved a reasonable score. Usage of Random Forests helps us to cope with the effect of the multiplicity of good models depending on varying subsets of features in the purchased items prediction and, in its turn, boosting is used as a suitable technique to overcome severe class imbalance of the buy-session prediction.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Filtering; 1.2.6 [**Artificial Intelligence**]: Learning

Keywords

supervised learning, class imbalance, e-commerce

1. INTRODUCTION

For us, the most challenging part of the competition is the classical problem in machine learning, *class imbalance* [**?**]. The task of the challenge is to predict "buy sessions" from a large set of sessions. Each session contains the click data of several anonymous users. However, only 5% of the training data consist of buy sessions, which is actually half a million among 9.3 million sessions. Hence, if we consider two classes (*buy* and *non-buy*), a severe class imbalance problem appears. Apart from predicting the buy sessions, the challenge organizers also want to know which items would be bought from a buy session. In order to answer these two questions, we have come up with two machine learning models, where one model is able to select the buy sessions and the other one finds the related items. For finding buy sessions while handling the class imbalance problem, we use the Adaboost.M1 implementation from Weka machine learning framework. We have finally found interesting features and achieved a reasonable score of 45,027 in the challenge.

2. PROBLEM STATEMENT

RecSys Challenge 2015 provides a data set containing the clickstream data of several users of an e-commerce site over the duration of six months. The training data set is composed of a set of sessions S and each session $s \in S$ contains a set of items I_s. $B_s \subseteq I_s$ denotes the set of items which has been bought in session s. There are two types of sessions S_b (the sessions that end in buying) and S_{nb} (the sessions that do not end in buying) as follows:

$$S_b = \{s \in S : B_s \neq \emptyset\}, \text{ and } S_{nb} = S \setminus S_b.$$

The test data also contains a set of sessions S_t and $S_t \cap S = \emptyset$. Now, given the set of sessions S_t, our task is to find all the sessions $S_{tb} \subseteq S_t$, which have at least one buy event. Moreover, if a session s contains a buy event, we have to predict the set of items B_s that was bought. The final solution should contain sessions Sl and the set of items A_s for each session $s \in Sl$, which should be bought. There is an original solution file, which contains a set of sessions S_b, and for each session $s \in S_b$ there is a set of items B_s which were bought in that session. However, rather than maximizing any traditional evaluation function we will have to maximize a unique scoring function based on the original solution file, which is shown below:

$$score(S_t) = \sum_{\forall s \in Sl} (-1)^{[s \notin S_b]} \frac{|S_b|}{|S_t|} + [s \in S_b] \frac{|A_s \cap B_s|}{|A_s \cup B_s|}, \text{ where}$$

$[Z]$ equals 1 if Z is true and 0 otherwise (Iverson notation).

2.1 Observation about the Scoring Function

According to the challenge description, buying event occurs in only 5% of the sessions and that's why we can assume that the penalty for predicting wrong sessions is only 0.05. From the other hand, for predicting a right session it is possible to achieve 1.05 score and as a consequence the strategy should be to identify most of the sessions as *buy sessions*. However, there is a limitation in the solution file size (<25MB) and that is why a proper level of precision should be reached in choosing the resulting sessions.

2.2 Data Description

The task contains three data files: click data file, buy data file and test data file. The click data file contains all the click sessions (S_c) and some information about the items clicked in those sessions, while the buy data file contains a set of sessions (S_b) in which at least one item has been bought. The click data file contains a set of tuples in the form {*sessionId, timestamp, itemId, category*} and the buy data file contains a set of tuples in the form {*sessionId, timestamp, itemId, price, quantity*}. The test data file follows the same data schema as the click data file. The data description is provided in Table 1.

Table 1: Data Description

File(s) type	Attributes	Description
click and buy	sessionId	Unique session id
click and buy	itemId	Unique item identifier
click and buy	timestamp	Time of click event or buy event for an item. It features day, month, hour and time of the specific event.
click file	category	The category of an item in a session. 0 indicates unknown category and $1, \ldots, 12$ denote regular categories. Other numbers indicate a brand, while "S" is the item from a special category.
buy file	price	Price of an item (0 when the price is not available)
buy file	quantity	Number of times a particular item has been bought in that session

3. DATA PREPROCESSING

3.1 Resolving Missing Category Information

At first we extract several properties of items and assign 0 as category number to the items with unknown category (almost half of the sessions items belong to this category). In fact the data were collected over six months period – from April to September, where most of the category information appeared after mid of June. We assume that the e-commerce site could not provide category data for that period of time. However, as an item repeatedly occurred in different sessions, it is possible to recover the category for most of the items using the data from June to September. While resolving the category we try to recover only regular category, i.e. we do not use special category . However, from our analysis it was evident that an item belongs to several categories. Nonetheless, we want to find a specific category for a specific item. So, we resolve item categories using the following rule: *if an item belongs to several categories, the actual category is its most frequent one in the click data.* After resolving the categories we append the attribute originalCategory to each click data.

3.2 Performing Temporal Ordering

At first, we sort both the click file and buy file using sessionId, which has a specific benefit that we will discuss later. Then for each of the sessions belonging to the files we sort the session data by timestamp. By so doing we can find the temporal ordering in the click and buy data. This temporal ordering helps us determine the order of user's clicks on the items in a session. Moreover, the duration of a click could easily be found by simply subtracting time of that click from the time of the next click. Now, for each distinct item in a session if we sum the duration of the clicks in which the item appears, we define the duration of the item in that session. After sorting by timestamp we append itemDuration (the amount of time an item is inspected in a session) to each click data.

3.3 Extraction of Item Properties

We extract several other properties, which are specific to an item and append it to each click data. The properties are listed in Table 2. At first we tried to solve the problem using only click-buy ratio and obtained a score of 29000. In that process, we averaged the click-buy ratio of all the items in a session and if the average was over the threshold of 5.5 we identified that session as a buy one. After that we picked half of the items by sorting on click-buy ratio. Using this simple approach we understood that we need to build features using this valuable statistic for developing our machine learning solution.

Table 2: Item Specific Properties

Item property	Description
click-buy ratio	buyCount/clickCount
clickCount	The number of buy sessions for an item. For clickCount=0 we increase it by 1
buyCount	The number of click sessions for an item. For buyCount=0 we increase it by 10
price	The price of an item. For price=0 we increase it by 1000.
itemDuration	The total time spent on an item over all the sessions in training set

Table 3: Data Properties

Statistics of training data	Description
Number of buy sessions (n_b)	509696
Number of non-buy sessions (n_c)	8740001
Distinct (item, buy session) pairs	1049817
Distinct (item, non-buy session) pairs	25565682
Number of items in training data	52146

3.4 Development of an Alternative Testbed

We have developed a complete framework in Java for performing the task, which includes the development of our own testbed. The development of our own independent testbed is crucial since there are only three chances of submission in the challenge per 24-hours. In order to do that we put each buy session s from all the sessions in our click data S_c to a file *clickBuy* (click sessions ended in buy) and all non-buy sessions are placed to the file *onlyClick* (click sessions not ended in buy). We sort all the click and buy sessions by sessionId in advance and split them finally (within $O(|S_c|log|S_c|)$).

After splitting, we randomly take half of the sessions in the clickBuy file to create our local test set and solution file; the remaining half of the data from clickBuy file is left for training our model and its evaluation in our own testbed.

Finally, we add randomly chosen one fourth of sessions from the onlyClick file to our new test set; the remaining goes to our new training set. As a result, we have our own original testing file and test (solution) file using which we can estimate the performance of our classifier. The number of sessions in our own test data is 2439481, while the original test data contains 2312432 sessions. Moreover, we kept 254510 buy sessions in our test data and according to scoring function shown in Section 2, we can achieve at most $254510 \times 1.05 = 267235.5$ score from our own test data.

4. PROPOSED SOLUTION

The task is intuitively divided in to two subtasks: predicting the outcome of a session and given a session predicting the set of items that should be bought in that session. So, we construct two classifiers to address these two subtasks. However, at first we thought of building a single classifier that would extract features from the items from the sessions in training data, and, given the items of a session in test data, it should classify the item as *buy* or *non-buy*. Unfortunately, in this process we have to handle a large feature data. If we look at Table 3, it is evident that developing only an item-based classifier would require to build a model for 1049817 items labeled with buy and 25565682 items marked as non-buy. Building any sophisticated classifier around that amount of data would take a huge computation time. Hence, at first we predict the sessions, which would end in buy, and then look for the prospective bought items in those sessions. So, we need to develop two classifiers: *item classifier* and *session classifier*.

4.1 Item Classifier

An important observation about item classifier is that we have to train the classifier only with the click data of the buy sessions in training data. The click data of a buy session contain a set of items that was bought (B_s) and a set of items that was not bought (A_s). Now for each item $i \in B_s$ we extract both session-based and item-based features. After that we label that item as *buy* and each item $i \in A_s$ as *non-buy*. By considering only the buy sessions we get 1049817 bought items and 1264870 non-bought ones. As a result, the class imbalance problem is less stressed and the classifier has to be trained with only 215053 data in total, which results in short training time.

Since we obtained our own testbed (see Section 3.4), we have our local solution file, which we can use to verify the performance of our method. We have tried to predict items with the classifier given the sessions in the local solution file i.e. we pretend that we know the correct sessions beforehand and we only try to test the efficiency of our item classifier. The maximum achievable score from our local solution file is 267235.5. Since we have met Rashomon effect (multiplicity of good subsets of features/models) in this item classification, we use Random Forests (RF) to cope with this difficulty [**?**]; moreover, RF classifier has shown the best score in our testbed and the highest Average Jaccard measure (see Table 5). The performance of the classifiers in shown in 5 and the selected features are described in Table 4. In order to extract features 9,10,14,15,21 and 21 we cluster session data using category that we resolved previously in 3.1.

Table 4: Item Features

No.	Feature name and description
1	click-buy ratio: click-buy ratio of an item
2	numberOfAppearance: number of times an item appeared in a session
3	itemPosition: the position of an item in the session after temporal ordering
4	isSunday: true if the item's session occurred on Sunday
5	isTuesday: true if the item's session occurred on Tuesday
6	hour: hour of the item's session
7	numberOfItems: number of distinct items in the item's session
8	itemAppearanceOverThree: number of items in a session appearing more than thrice in it
9	isFirstItemCategory: true if the item is the first one clicked in a specific category of the session.
10	isLastItemCategory: true if the item is the last one clicked in a specific category of the session.
11	buyCount: number of sessions in training data where the item is bought
12	itemDuration: number of seconds spent on the item in the specific session
13	price: price of the item
14	categoryTopClickBuyRatio: 1 if the click-buy ratio of the item is the highest within its category and session, otherwise 0
15	categoryTopBuy: 1 if the buyCount of the item is the highest within its category and session, otherwise 0
16	sessionTopClickBuyRatio: 1 if the click-buy ratio of the item is the session highest, otherwise 0
17	sessionTopBuyCount: 1 if the buyCount of the item is the highest in its session, otherwise 0
18	maxDuration: 1 if the duration of the item is the highest in its category and session, otherwise 0
19	itemOwnDuration: is the period of time that the item has been observed in the training data
20	clickCount: number of sessions in training data where the item has been clicked
21	categoryLowestPrice: 1 if the price of the item is the lowest within its category and session, otherwise 0
22	categoryHighestPrice: 1 if the price of the item is the lowest in its category and session, otherwise 0

Table 5: Item Classifier Performance

Classifier	Score	Possible score	Average Jaccard	Time to build
Naïve Bayes	172211	267235.5	0.699	**16 s**
Bayes net	182226	267235.5	0.727	65 s
Logistic regression	186561	267235.5	0.751	123 s
Random forest	**189537**	267235.5	**0.767**	218 s

4.2 Session Classifier

We have 509696 buy sessions and 8740001 non-buy ones in the training data. Rather than looking at the feature of the items, we look at the features of sessions. There are

two important observations for building the session classifier: 1) Since we have only 0.05 penalty for selecting a non-buy session, it should be a high recall classifier; 2) As there exists class imbalance problem, a classifier would tend to predict most of the sessions as *non-buy*.

In order to address the problems we performed resampling of data and classified sessions using AdaBoost.M1 algorithm. AdaboostM1 comes as a remedy for class imbalance problem combined with resampling [**?**], and it gives us the highest recall among the classifiers we tested. Table 8 shows that even though RF works better when classifying items, it cannot reach high recall when classifying sessions; moreover, it takes the highest time for model training. For Adaboost.M1 the model is built in a smaller period of time (3 min 2 s for 1896886 feature vectors) and evaluation is done in 23 s for 2312432 feature vectors. We have experimented with Adaboost.M1 using different set of features; we describe the selected features in Table 6. In Table 7 we show the performance of Adaboost.M1 with the set of all the extracted features and it produces the highest score – 107763.7. However, in our local testbed it returns a huge number of sessions that cannot be accommodated in the solution file of less than 25 MB (the challenge constraint). From Table 7 one can observe that with selected features we obtain a score of 106508.4, which is the highest considering the file size. Finally, we have obtained our best challenge score 45027 by cascading Adaboost.M1 with resampling for session classification and RF for choosing items.

In Table 7 we show the result of considering different subsets of features obtained from Table 6. We excluded the time based features, aggregated features, and other subsets to evaluate the session classifier. Moreover, we mention two of the challenge scores and show the respective score from our own testbed. As there are 2312432 session in the challenge test data, we can assume that there are 115622 (5%) buy sessions in it and the Maximum Possible Score (MPS) from the test data can be 115622 $\times$ 1.05 =121402, approximately. However, we kept 254510 buy sessions in our test data and the MPS from our testbed can be 267235.5 (see Table 5). As shown in Table 7, we achieve our best score 45027 (37% of challenge test data), when our testbed score is 106508 (39% of our own test data). When we achieve 39127 (32% of challenge test data), our testbed score is 96077 (35% of our own test data). So, we can assert that we are able to approximate our challenge score from our testbed one.

5. CONCLUSIONS

It seems the proposed approach can be applied in a similar clickstream classification setting. The usage of a cascade of two different classifiers is beneficial when one experiences severe imbalance between buy and non-buy sessions and multiplicity of good feature subsets. Moreover, learning and classification phases are reasonably short for this subtask decomposition. We have no doubts that further model tuning would definitely allow to achieve a better score.

Table 6: Selected Session Features

No.	Feature name and description
1	numberOfClicks: number of clicks in a session
2	numberOfItems: number of distinct items in a session
3	itemsOverClickBuyRatio: number of items having click-buy ratio over 3.6, median click-buy ratio of all items
4	itemsOverClickBuyCount: number of items having click-buy count over 57, average buyCount of all items
5	averageClickBuyRatio: average click-buy ratio over all the items in a session
6	averageBuyCount: average buy count for all the items in a session
7	itemAppearanceOverTwo: number of items in a session appearing more than twice in it
8	itemAppearanceOverThree: number of items in a session appearing more than three times in it
9	duration: duration of a session
10	hour: hour of the day when the session occurred
11	isSunday: true if the session day is Sunday
12	isTuesday: true if the session day is Tuesday
13	averageItemDuration: itemDuration for an item is the amount of time spent on that item in the training data. This feature is the average over itemDuration of each item in a session
14	averageItemClickCount: clickCount for an item is the number of times an item has been clicked in the training data. This feature is the average of clickCount of each item in a session
15	averageItemPrice: average item price in a session

Table 7: Feature Selection for Session Classifier

Selected features	R	P	F_1	Overall score	Final score	Number of sessions
all (>25MB)	**0.77**	0.32	0.35	**107764**	n/a	**963307**
selected	0.71	0.23	0.34	106508	**45027**	782337
w/o time-based	0.60	**0.27**	**0.37**	88445	n/a	563443
w/o 3 and 4	0.72	0.22	0.36	103300	n/a	832275
1,5,6,7,15	0.67	0.22	0.34	96077	39127	765873
w/o 1 and 2	0.69	0.24	0.35	103944	n/a	744994
w/o aggregated	0.66	0.25	0.36	98011	n/a	684560

Table 8: Session Classifier Performance

Classifier	Score	Possible score	R	P	Model building
Naïve Bayes	49890.8	267235.5	0.36	**0.30**	**30 s**
Random Forest	87854.6	267235.5	0.58	0.26	1308 s
Bayes Net	90865.5	267235.5	0.66	0.23	146 s
Logistic Regression	100134.6	267235.5	0.67	0.25	152 s
Adaboost.M1	**106508.3**	267235.5	**0.71**	0.23	178 s